

**Source Code Readability Improvement
Using Heuristic-Based
Dynamic Error Reporting
During Editing**

Phillip Anthony Relf

Doctor of Philosophy in Engineering

2007

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student

Acknowledgement

I thank my supervisors David Lowe and John Leaney for their patient direction and pertinent review comments, and particularly early on for their restraint when I was struggling to understand the full implications and the nature of the research process. I owe these two gentlemen much for their gentle direction in steering my paradigm shifts that have occurred during the prior years. In addition, I thank the following people for their support during my research: Susan Cockshell for reviewing and offering advice on questionnaire layout and wording, and also for Human Factors advice relating to the instructions for the experiment test subjects; Chris Coreless for removing the Java syntax errors from my translation of the original Ada experiment source code files into the Java computer language; the 2005 UTS classes: Software Systems Analysis, Software Systems Design and Software Analysis & Design, and also to Mark Denford and Jared Berghold for their participation as experiment test subjects. I also acknowledge the participation of the many corporate programmers who responded to questionnaires and supplied data from their participation in the experiment, and in particular to Anthony Anger, Chris Coreless, Martin Fowel, Grant Hensley, Ken Lau, Patrick Lee, Sun-Young Kim and Amir Sadross for their participation in the case studies. Finally, I thank my employer for allowing me to collect and report data as part of the industrial case studies.

I thank those anonymous reviewers who supplied comment on my research papers.

I wish to again acknowledge David Lowe for supplying editorial input.

For my parents: Dorothy, Gertrude, Ron and Sid.

Table of Contents

1	Introduction	1
1.1	Research Problem.....	1
1.2	Research Overview	4
1.3	Research Objectives	5
1.4	Research Publications	5
1.5	Document Structure	6
1.6	Chapter Summary.....	8
2	Background Theory.....	9
2.1	Software Maintenance.....	9
2.1.1	Inevitability of Software Maintenance.....	9
2.1.2	Size Related Difficulties of Software Maintenance	10
2.1.3	Software Process Scaling	12
2.1.4	Source Code Readability and Software Maintenance.....	13
2.1.5	Economic Case for Software Maintainability	15
2.2	Software Quality	16
2.2.1	Software Quality Practice	17
2.2.2	Software Quality Categorisation.....	18
2.2.3	Software Quality Improvement.....	19
2.2.4	Manual Source Code Inspection	21
2.2.5	Automated Source Code Inspection.....	22
2.2.6	Identifier-Naming Style Standardisation.....	23
2.3	Programmer.....	24
2.3.1	Programmer's Education.....	24
2.3.2	Programmer's Cognitive Limitations.....	26
2.3.3	Management's Culture	27
2.3.4	Programmer's Culture.....	28
2.4	Chapter Summary.....	29
3	Research Method.....	31
3.1	Foreground Theory.....	31
3.1.1	Hypothesis Statement.....	32
3.1.2	Hypothesis Verification	33
3.2	Research Questions	34
3.2.1	Identifier-Naming Style and Software Quality	34
3.2.2	Catalogue of Identifier-Naming Style Guidelines.....	34
3.2.3	Cognitive Complexity of Identifier-Naming Style Guidelines	36
3.2.4	Detection of Identifier-Naming Style Flaws	42
3.2.5	Automation of Identifier-Naming Style Guidelines.....	43
3.2.6	Perceptions of Identifier-Naming Style Flaws.....	43
3.2.7	Display of Identifier-Naming Style Flaws	44
3.2.8	Benefit of Reporting Identifier-Naming Style Flaws.....	45
3.2.9	Effect of Reporting Identifier-Naming Style Flaws.....	47
3.2.10	Suggestion of Replacement Identifier Name	48
3.2.11	Effect of Suggested Replacement Identifier Name.....	48
3.2.12	Production Software Identifier-Naming Style Flaws.....	49
3.2.13	Measurement of Identifier-Naming Style Flaws.....	50
3.2.14	Source Code Readability Measurement	50
3.2.15	Research Question Summary Findings	52

3.3	Research Variables.....	56
3.3.1	Dependent Variable Measurement.....	57
3.3.2	Professional Programmer.....	57
3.3.3	Years Programming.....	58
3.3.4	Large Program Experience.....	58
3.3.5	Code Review Experience – Subject.....	59
3.3.6	Coder Review Experience – Reviewer.....	59
3.3.7	Touch Typing Ability.....	59
3.3.8	Identifier Naming Effort.....	60
3.3.9	Flexible Work Practices.....	60
3.4	Literature Review Method.....	60
3.5	Programmer Characteristics Questionnaire.....	63
3.5.1	Questionnaire Statement Wording.....	64
3.5.2	Questionnaire Presentation.....	66
3.6	Identifier-Naming Style Guideline Programmer Acceptance Questionnaire.....	67
3.6.1	Questionnaire Statement Wording.....	68
3.6.2	Questionnaire Presentation.....	69
3.6.3	Questionnaire Respondent Selection Criteria.....	71
3.7	Research Metrics.....	72
3.7.1	Cyclomatic Complexity.....	72
3.7.2	Software Science Metrics.....	73
3.7.3	Readability Predictor.....	74
3.7.4	Maintainability Index.....	76
3.8	Source Code Editor.....	77
3.8.1	User Interface.....	77
3.8.2	Dynamic Reporting.....	79
3.8.3	Identifier Name Replacement Suggestion.....	80
3.8.4	Design Considerations.....	81
3.8.5	Session Log.....	82
3.9	Identifier-Naming Style Flaw Analysis and Report Generator.....	83
3.10	Textbook Survey.....	83
3.11	Production Software Survey.....	85
3.11.1	Contemporary Software Survey.....	85
3.11.2	Dated Software Survey.....	86
3.12	Maintenance and Production Experiment.....	86
3.12.1	Introduction Exercise.....	87
3.12.2	Maintenance Exercise.....	87
3.12.3	Production Exercise.....	88
3.12.4	Test Subject Selection.....	89
3.12.5	Experiment Delivery.....	89
3.12.6	Test Subject Instruction.....	91
3.12.7	Statistical Tests.....	92
3.13	Case Studies.....	93
3.13.1	Task Specification.....	93
3.13.2	Task Staffing.....	95
3.13.3	Novice Programmer Case Study Method.....	95
3.13.4	Programming Team Case Study Method.....	97
3.13.5	Data Collection and Analysis.....	97
3.14	Research Ethics.....	98
3.15	Chapter Summary.....	99

4	Identifier-Naming Style Guidelines Specification	101
4.1	Literature Review Results	101
4.1.1	Single Identifier – Character Relationships	101
4.1.2	Single Identifier – Character Count Relationships.....	103
4.1.3	Single Identifier – Word Count Relationships	105
4.1.4	Single Identifier – Word Qualification Relationships.....	106
4.1.5	Single Identifier – Word Meaning Relationships.....	107
4.1.6	Single Identifier – Naming Convention	110
4.1.7	Multiple Identifier Relationships	112
4.1.8	Identifier Name Natural Language Meaning	113
4.2	Identifier-Naming Style Guideline Implementation	115
4.3	Suggested Replacements	115
4.4	Chapter Summary.....	121
5	Investigation & Results	122
5.1	Identifier-Naming Style Guideline Programmer Acceptance Questionnaire.....	122
5.1.1	Attitude Statement Acceptance	122
5.1.2	Attitude Statement Agreement.....	124
5.1.3	Novice/Expert Comparisons	127
5.2	Textbook Survey	128
5.3	Contemporary Software Survey.....	130
5.3.1	Ada and Java Software Comparison	132
5.3.2	Computer Program Size	133
5.4	Dated Software - Survey	135
5.4.1	Temporal Comparison.....	136
5.4.2	Identifier-Naming Style Flaw Trends	138
5.4.3	Single Programmer Sample Concerns	139
5.5	Identifier-Naming Style Guideline Validity.....	140
5.6	Source Code Editor	146
5.7	Programmer Characteristics Questionnaire.....	148
5.8	Maintenance and Production Experiment.....	151
5.8.1	Expert Programmer Identifier Name Adjudication.....	151
5.8.2	Control/Experimental Group Differences	153
5.8.3	Readability Metrics	154
5.8.4	Introduction Exercise	155
5.8.5	Maintenance Exercise	157
5.8.6	Production Exercise	160
5.8.7	Software Bugs	162
5.8.8	Acceptance of Suggested Replacement Identifier Names	163
5.8.9	Maintenance and Production Experiment Completion Time.....	164
5.8.10	Experiment Overview	165
5.8.11	Confounding Variables	165
5.9	Novice Programmer Case Study	169
5.9.1	Novice Programmer Comments.....	170
5.9.2	Source Code Inspection.....	171
5.9.3	Source Code Effects.....	172
5.10	Programming Team Case Study.....	173
5.10.1	Source Code Editor Usage	174
5.10.2	Source Code Inspection.....	174
5.10.3	Reporting Effects on Source Code.....	175
5.11	Identifier-Naming Style Guideline Support.....	178

5.12	Chapter Summary.....	184
6	Conclusions	187
6.1	Summary of Findings	187
6.1.1	Importance of Identifier Naming	187
6.1.2	Identifier-Naming Style Guideline Summary	188
6.1.3	Temporal Effects on Identifier-Naming Style.....	190
6.1.4	Computer Programming Language Effects on Identifier-Naming Style 191	
6.1.5	Effort Required to Chose Meaningful Identifier Names.....	192
6.2	Limitations in Material.....	192
6.2.1	Identifier-Naming Style Guideline Limitations	192
6.2.2	Identifier-Naming Style Guideline Implementation Limitations	193
6.2.3	Identifier-Naming Style Guideline Programmer Acceptance Questionnaire Limitations.....	193
6.2.4	Textbook Survey Limitations.....	195
6.2.5	Dated Software Survey Limitations	195
6.2.6	Contemporary Software Survey Limitations	196
6.2.7	Source Code Editor Limitations.....	196
6.2.8	Programmer Characteristics Questionnaire Limitations	197
6.2.9	Maintenance and Production Experiment Limitations.....	197
6.2.10	Novice Programmer Case Study Limitations.....	197
6.2.11	Programming Team Case Study Limitations	198
6.3	Research Contribution.....	198
6.3.1	Research Relevance	199
6.3.2	Software Maintenance Legacy	199
6.3.3	Confounding Variables	200
6.3.4	Software Development Support	200
6.3.5	Software Engineering Teaching.....	201
6.4	Research Hypothesis Discussion.....	202
6.4.1	Basic Research Proposition Discussion	202
6.4.2	Research Hypothesis Parts Discussion	203
6.4.3	Research Conclusions	205
6.5	Further Work.....	206
6.5.1	Identifier-Naming Practices	207
6.5.2	Identifier-Naming Standardisation.....	208
6.5.3	Software Engineering Practice.....	209
6.6	Chapter Summary.....	209
Appendix A -	Identifier-Naming Style Guidelines.....	A-1
A.1	Un-named Constant	A-2
A.2	Multiple Underscore	A-3
A.3	Outside Underscore	A-3
A.4	Numeric Digit(s).....	A-4
A.5	Short Name	A-4
A.6	Long Name	A-5
A.7	Word Count	A-5
A.8	Identifier Encoding	A-6
A.9	Class/Type Qualification	A-8

A.10	Constant/Variable Qualification	A-9
A.11	Abstract Words	A-10
A.12	English Words	A-11
A.13	Numeric Name.....	A-12
A.14	Plural Word	A-13
A.15	Naming Convention.....	A-14
A.16	Duplicate Names	A-15
A.17	Similar Names	A-16
A.18	Unused Identifier	A-17
A.19	Same Words.....	A-18
Appendix B -	Identifier-Naming Style Guideline Programmer Acceptance Questionnaire B-1	
Appendix C -	Programmer Characteristics Questionnaire	C-1
Appendix D -	Test Subject Ethics Release & Instruction	D-1
Appendix E -	Experiment Files.....	E-1
Appendix F -	Source Code Editor Software Modules	F-1

List of Tables

Table 3-1 - Programmer Cognitive Effort.....	38
Table 3-2 - Research Question Investigation Summary	53
Table 3-3 - Electronic Search Result Breakdown	63
Table 3-4 - Cyclomatic Complexity Conversion	73
Table 3-5 - Maintainability Index Conversion.....	76
Table 5-1 - Attitude Statement Pearson Correlations	123
Table 5-2 - Identifier-Naming Style Guideline Programmer Acceptance Questionnaire Coding Means	125
Table 5-3 - Non-Accepted Identifier-Naming Style Guideline Id	126
Table 5-4 - Computer Programming Language Course Textbooks	128
Table 5-5 - Course Textbook Page Numbers.....	129
Table 5-6 - Identifier-Naming Style Flaws in Contemporary Software	132
Table 5-7 - Identifier-Naming Style Flaws by Computer Program Size Range	134
Table 5-8 - Identifier-Naming Style Flaws in Dated Software	137
Table 5-9 - Group Ada/Java and Individual Pascal Software Project Correlations	140
Table 5-10 - Identifier-Naming Style Guideline Summary	141
Table 5-11 - Programmer Characteristics	149
Table 5-12 - Introduction Exercise Correlation between Expert Programmer Identifier Name Adjudication	152
Table 5-13 - Readability Metric Correlations	155
Table 5-14 - Introduction Exercise Results.....	156
Table 5-15 - Maintenance Exercise Results.....	157
Table 5-16 - Failed Maintenance Actions.....	158
Table 5-17 - Production Exercise Results.....	161
Table 5-18 - Software Bugs and Meaningful Identifier Names	163
Table 5-19 - Maintenance and Production Experiment Times	164
Table 5-20 - Confounding Variable Support Findings	166
Table 5-21 - Novice Programmer Source Code Inspection Review Comments Summary	171
Table 5-22 - Programming Team Source Code Inspection Review Comments	175
Table 5-23 - Background Theory Support Findings	179

List of Figures

Figure 2-1 - Minimum SLOC Counts for Large Computer program	12
Figure 3-1 - Programmer Characteristics Questionnaire Presentation.....	67
Figure 3-2 - Programmer Acceptance Questionnaire Presentation.....	70
Figure 3-3 - Source Code Editor User Interface	78
Figure 3-4 - Identifier-Naming Guideline Activation List Dialog Box	80
Figure 5-1 - Identifier-Naming Style Flaw Reduction by Time	138
Figure 5-2 - Identifier-Naming Style Flaw Trends	139
Figure 5-3 - Average Percentage of Meaningful Identifier Names.....	154
Figure 5-4 - Identifier-Naming Style Flaw Reduction Comparison between Programmers	176

Abstract

This research considers whether dynamically reporting poor identifier-naming practices at the time when the source code is written can improve readability and hence maintainability. Poor identifier-naming practices have little effect on the production phase of the software lifecycle. However, poor identifier-naming practices can have a substantial impact during the maintenance phase of the software lifecycle, particularly for the maintenance of large (i.e., 1M SLOC) computer programs. Of the nineteen identifier-naming style guidelines employed to support the research and used to identify poor identifier-naming practices, thirteen were found to be useful in improving source code readability. A questionnaire was employed to ascertain whether expert programmers accepted these guidelines; a textbook survey was used to identify the potential to transmit poor identifier-naming practices; a survey of contemporary source code was used to ascertain current identifier-naming practices; and a survey of dated source code was used to ascertain how identifier-naming practices have changed over an extended period of time. In addition, a controlled experiment was used to evaluate the effects of poor identifier-naming during a maintenance exercise and to evaluate the generation of poor identifier-naming during a production activity. A novice programmer case study and a programming team case study were executed to identify the longer term effects of dynamically reporting poor identifier-naming practices. The benefit of dynamically reporting poor identifier-naming practices was most pronounced for novice programmers with the percentage of meaningful identifier names increasing from 12% to 28%. The results for expert programmers were less pronounced with the percentage of meaningful identifier names correspondingly increasing from 53% to 60%. The identifier-naming style guidelines that proved to be the most useful to programmers required that identifier names should be composed of from two to four Natural language words or project accepted acronyms; should not be composed only of abstract words; should not contain plural words; and should conform to the project naming conventions.

List of Abbreviations and Acronyms

ACM	Association for Computing Machinery
API	Application Programming Interface
CATS	Computer Architecture Topography Simulator
CD	Compact Disk
COCOMO	Constructive Cost Model
df	Degrees of Freedom
DoD	Department of Defense
Exp.	Experiment
GUI	Graphical User Interface
Id	Identifier
IDE	Interactive Development Environment
IEEE	Institute of Electrical and Electronic Engineers
Lab.	Laboratory
LOC	Line of Code
N/A	Not Applicable
PC	Personal Computer
ROM	Read Only Memory
SDD	Software Design Description
SLOC	Source lines of Code
SPC	Software Process Consortium
SRS	Software Requirements Specification
UML	Unified Modelling Language
USA	United States of America
UTS	University of Technology, Sydney
WYSIWYG	What You See Is What You Get
Y2k	Year 2000

Glossary of Terms

Most technical terms in this thesis are used consistently with the definitions given in IEEE Std 610.12-1990: *IEEE Standard Glossary of Software Engineering Terminology*. Additional terms used within the thesis, but which are not given in this standard are defined below.

Mission Critical

Mission Critical, when applied to a computer system, defines that computer system as being critical to the successful completion of the mission.

SLOC

The term SLOC is potentially ambiguous when used to identify the size of a computer program, in that the count of source lines of code can mean: (1) the total number of source lines of code generated during the production of the computer program; (2) the total number of source lines of code delivered to the customer; or (3) the total number of source lines of code necessary to build the computer program, excluding any support software. The last usage of the term SLOC hence excludes any SLOC counts attributed to test harness software and test program software. This last usage of the term SLOC has been assumed and is used within the body of this thesis.